

# Existence of k-ary Trees: Subtree Sizes, Heights and Depths

Akshar Varma<sup>1</sup>

- 1 Dhirubhai Ambani Institute of Information and Communication Technology,  
Gandhinagar, India  
akshar\_varma@daiict.ac.in

---

## Abstract

The rooted tree is an important data structure, and the subtree size, height, and depth are naturally defined attributes of every node. We consider the problem of the existence of a k-ary tree given a list of attribute sequences. We give polynomial time ( $O(n \log(n))$ ) algorithms for the existence of a k-ary tree given depth and/or height sequences. Our most significant results are the Strong NP-Completeness of the decision problems of existence of k-ary trees given subtree size sequences. We prove this by multi-stage reductions from NUMERICAL MATCHING WITH TARGET SUMS. In the process, we also prove a generalized version of the 3-PARTITION problem to be Strongly NP-Complete. By looking at problems where a combination of attribute sequences are given, we are able to draw the boundary between easy and hard problems related to existence of trees given attribute sequences and enhance our understanding of where the difficulty lies in such problems.

**1998 ACM Subject Classification** F.2.2 [Analysis of Algorithms and Problem Complexity] Non-numerical Algorithms and Problems—Computations on discrete structures; G.2.2 [Discrete Mathematics] Graph Theory—Trees  
General Terms: Theory, Algorithms

**Keywords and phrases** Existence of k-ary Trees, Strong NP-Completeness, Subtree Sizes, Height, Depth

## 1 Introduction

Rooted trees are important data structures that are encountered extensively in Computer Science, especially in the form of self-balancing binary trees. Attributes of nodes, like its subtree size, height, and depth are invariant under isomorphism of rooted trees and are ubiquitous in the study of data structures and algorithms. Heights are used in self-balancing trees (AVL [2] or Red-Black [10]), depths in analyzing complexity of computation trees, recursion trees and decision trees and subtree sizes for finding order statistics in dynamic data sets [5]. The subtree size, height and depth of every node in a rooted tree can be computed in time linear in the number of nodes. In this paper, we discuss the computational complexity of the converse problem – the existence (realization) of a rooted k-ary tree given some of these attribute sequences.

The problems that we address are similar in flavor to those studied in Aigner and Triesch [3] who discuss realizability and uniqueness of graphs given invariants. The most famous of such problems is the Erdos-Gallai graph realization problem [7] (a variant also addresses the realization problem for trees) which asks whether a given set of natural numbers occur as the degree sequence of some graph; polynomial time algorithms are known for this problem [15, 11]. Our problem can be considered a part of the category of well researched problems of reconstruction of a combinatorial structure from some form of partial information.

Apart from the Erdos-Gallai theorem we already mentioned, a lot of work has been done on reconstruction of graphs from subgraphs [23, 16, 12, 13, 14, 21, 17, 19]. Beyond graphs, the problem of reconstruction of combinatorial structures like matrices [20] and trees [18] have also been studied. More recent work has been done on reconstruction of sequences [6] and on reconstruction of strings from substrings [1]. Bartha and Bursci [4] have addressed the problem of reconstruction of trees using frequencies of subtree sizes. While their paper focuses on the reconstruction of unrooted trees given subtree sizes, we look at the existence of rooted trees with given attribute sequences.

Given a rooted tree  $T$ , information  $I(T)$  about the attributes of the tree  $T$  can be constructed using various combination of attribute sequences. Given some such information  $I$ , we look at the existence problem  $E(I)$ , which asks whether there is a  $k$ -ary tree  $T$  such that  $I = I(T)$ . We use the letters  $S$ ,  $H$ , and  $D$  to refer to subtree size, height and depth attributes respectively. These attributes can be used individually or in combination as in these examples:

► **Example 1.1.** We use the notation like  $E(I_S)$  for the existence problem given only, say, the subtree sizes sequence<sup>1</sup>. For example: Given a sequence of subtree sizes,  $I_S = \{1, 2, 3, 1, 1, 3, 7, 1, 9\}$ , does there exist a tree  $T$  such that  $I_S = I_S(T)$ ?

► **Example 1.2.** We use the notation like  $E(I_{S,D})$  to refer to the existence problem given, say, synchronized subtree sizes and depths (all attributes of a node are associated with each other as tuples). For example: Given synchronized information of (subtree size, depth),  $I_{S,D} = \{(1, 5), (2, 4), (3, 2), (1, 3), (1, 3), (3, 2), (7, 1), (1, 1), (9, 0)\}$ , does there exist a tree  $T$  such that  $I_{S,D} = I_{S,D}(T)$ ?

► **Example 1.3.** We use the notation like  $E(I_S, I_D)$  when there is no synchronization and just two (or more) sequences are given. For example: Given asynchronized information list of subtree sizes and depths,  $I_S = \{1, 2, 3, 1, 1, 3, 7, 1, 9\}$ ,  $I_D = \{5, 4, 2, 3, 3, 2, 1, 1, 0\}$ , does there exist a tree  $T$  such that  $I_S = I_S(T)$  and  $I_D = I_D(T)$ ?

Problems containing only height and/or depth sequences are shown to have  $O(n \log(n))$  algorithms for deciding the existence of  $k$ -ary trees. Our most significant results are the proof that all existence problems containing subtree size sequences are Strongly NP-Complete for  $k$ -ary trees. We prove the Strong NP-Completeness using reductions from the NUMERICAL MATCHING WITH TARGET SUMS problem. The reduction is performed in multiple stages during which we also prove a generalized version of the 3-PARTITION problem to be Strongly NP-Complete. We then proceed to provide slightly modified yet similar existence problems (for example, existence of certain sub-classes of trees) which are polynomially solvable. We attempt to draw the boundaries separating the NP-Complete problems from the easy problems, focusing on how changing the attribute, adding restrictions or providing more information change the computational complexity.

Section 2 contains the basic definitions, notation and conventions. The most important results are presented in Section 3.1, which contains the proofs for the Strong NP-Completeness of problems related to subtree sizes. Section 3.2 continues further discussion on subtree sizes and contains algorithms for some sub-classes of trees for which the problem can be solved

---

<sup>1</sup> We use the term sequence (borrowed from the terminology in the Erdos-Gallai theorem) throughout the paper since these attributes are generally computed and used in either non-decreasing or non-increasing order. It should be noted that when such an order is not imposed, these are multisets. Nonetheless, we maintain the use of the term sequence for consistency.

in polynomial time. This is followed by the Section 4, detailing the analyses of the height and depth sequences. Section 5 contains details of sequences given in combination and some discussion about the difficulty of these problems. Section 6 has concluding remarks and possible directions for future work.

## 2 Preliminaries

► **Definition 2.1** (k-ary Tree). A rooted tree in which every node can have at most  $K$  children.

► **Definition 2.2** (Subtree size). The number of nodes in the subtree rooted at a node (including itself) is known as the subtree size of that node. The subtree size of a node can also be defined recursively as being one greater than the sum of the subtree sizes of its children.

► **Definition 2.3** (Height of a node). The height of a leaf node is zero. The height of every other node is one more than the maximum of the heights of its children.

► **Definition 2.4** (Depth of a node). The depth of a node is the number of edges in the path from that node to the root.

► **Definition 2.5** (Levels in a tree). The set of nodes at a particular depth forms the level at that depth.

► **Definition 2.6** (Complete Trees).  $K$ -ary trees in which every level except possibly the last are filled and all nodes in the last level are filled from the left. Given the number of nodes  $n$ , this is a unique tree and is represented as  $T^c(n)$ .

► **Definition 2.7** (Full Trees).  $K$ -ary trees in which every node has exactly  $K$  or 0 children.

The remaining definition are due to Gary and Johnson [9, 8].

► **Definition 2.8** (MAX[I]). MAX[I] is the magnitude of the maximum number present in an instance  $I$  of any decision problem.

► **Definition 2.9** (LENGTH[I]). LENGTH[I] is the number of symbols required to represent an instance  $I$  of any decision problem.

► **Definition 2.10** (Strongly NP-Completeness). For a decision problem  $\Pi$ , we define  $\Pi_p$  to denote the subproblem of  $\Pi$  obtained by restricting  $\Pi$  to only those instances that satisfy  $\text{MAX}[I] \leq p(\text{LENGTH}[I])$ , where  $p$  is a polynomial function. The problem  $\Pi$  is said to be NP-Complete in the strong sense or Strongly NP-Complete (SNPC) if  $\Pi$  belongs to NP and there exists  $p$  for which  $\Pi_p$  is NP-complete.

► **Definition 2.11** (Pseudo-polynomial transformation). A pseudo-polynomial transformation from a source problem to a target problem is a transformation  $f$  from any instance of the source problem to an arbitrary instance of the target problem satisfying the following conditions:

- $f$  should be computable in a time polynomial in  $\text{MAX}[I]$  and  $\text{LENGTH}[I]$ .
- $\text{LENGTH}[f(I)] \leq p(\text{LENGTH}[I])$  for some polynomial  $p$ .
- $\text{MAX}[f(I)] \leq p'(\text{MAX}[I], \text{LENGTH}[I])$  for some polynomial  $p'$ .

► **Remark 2.12.** To prove problems to be Strongly NP-Complete, one needs to have a pseudo-polynomial transformation from a problem already known to be Strongly NP-Complete. In practice this is the same as a polynomial transformation used to prove problems NP-Complete, with the added condition that the maximum integer in the constructed instance needs to be polynomially bounded in the maximum integer in and the length of, the instance from which we are making the transformation.

### 3 Existence of k-ary Trees given Subtree Sizes Sequence: $E(I_S)$

Given a tree, finding the subtree sizes of all its nodes is a linear time problem. One might intuitively expect that the converse problem is also easy. After all, by definition, all that needs to be assured is that the sum of the children's subtree sizes is one less than the parent's subtree size. While intuitively this may seem so, the  $E(I_S)$  problem has been found to be difficult to be solve for binary trees [22]. Top-down, bottom-up and dynamic programming approaches were tried but all yielded exponential time algorithms. This difficulty prompted a search for an NP-Completeness reduction.

We prove the  $E(I_S)$  problem Strongly NP-Complete in a series of reductions starting from the NUMERICAL MATCHING WITH TARGET SUMS (NMTS) problem (Strongly NP-Complete by Theorem 3.1) to the NUMERICAL MATCHING WITH TARGET SUMS USING K-SETS (NMTS-K) problem (in Section 3.1.1) to the K-PARTITION WITH TARGETS (K-PWT) problem (in Section 3.1.2) to finally the  $E(I_S)$  problem (in Section 3.1.3).

#### 3.1 Proofs of Strong NP-Completeness

In this section we prove the Strong NP-Completeness of  $E(I_S)$  via a series of reductions.

► **Theorem 3.1** (Due to Garey and Johnson [9]). *The NMTS problem stated below is strongly NP complete:*

*Given disjoint sets  $X$  and  $Y$  each containing  $m$  elements, a size function  $s : X \cup Y \mapsto \mathbb{Z}^+$ , and a target vector  $B = (b_1, \dots, b_m) \in \mathbb{N}^m$  with positive integer entries, can  $X \cup Y$  be partitioned into  $m$  disjoint sets  $A_1, A_2, \dots, A_m$ , each containing exactly one element from each of  $X$  and  $Y$ , such that,  $\sum_{a \in A_i} s(a) = b_i$ , for  $1 \leq i \leq m$ ?*

NMTS  $(X, Y, s, B, m)$  refers to an instance of the NMTS problem characterized by the sets  $X$  and  $Y$ , a size function  $s$ , the target vector  $B$  and the cardinality of the target vector  $m$ .

##### 3.1.1 NMTS-K is Strongly NP-Complete

The NMTS-K problem is proved Strongly NP-Complete by reduction from NMTS.

► **PROBLEM 3.2** (NMTS-K). Given  $K \geq 2$  disjoint sets  $X_i$  each containing  $m$  elements, a size function  $s : \bigcup X_i \mapsto \mathbb{Z}^+$ , and a target vector  $B = (b_1, \dots, b_m) \in \mathbb{N}^m$  with positive integer entries, can  $\bigcup X_i$  be partitioned into  $m$  disjoint sets  $A_1, A_2, \dots, A_m$ , each containing exactly one element from each of  $X_i$ , such that,  $\sum_{a \in A_i} s(a) = b_i$ , for  $1 \leq i \leq m$ ?

NMTS-K  $(K, X_i, s, B, m)$  is an instance of the NMTS-K problem characterized by the integer  $K$ , the  $K$  sets  $X_i$ , a size function  $s$ , the target vector  $B$  and the cardinality of the target vector  $m$ .

**Proof.** The NMTS-K problem is in NP since given a candidate partition  $A_i$ , we only need to verify that,  $\sum_{a \in A_i} s(a) = b_i$ , for  $1 \leq i \leq m$ . We now construct an instance NMTS-K  $(K, X_i, s', B', m')$  of NMTS-K problem from an instance NMTS  $(X, Y, s, B, m)$  of the NMTS problem using the following transformation for  $K \geq 3$  since for  $K = 2$ , the NMTS-K problem is the NMTS problem. Note that this is a polynomial transformation since computing Equations 2, 3 and 4 can be done in polynomial time.

$$m' = m, X_1 = X, X_2 = Y \quad (1)$$

$$X_i \text{ are disjoint sets such that } |X_i| = m' \text{ for } 3 \leq i \leq K \quad (2)$$

$$s'(x) = \begin{cases} s(x) & x \in X \cup Y \\ 1 & \text{otherwise} \end{cases} \quad (3)$$

$$B' = (b'_1, b'_2, \dots, b'_m) \text{ where } b'_i = b_i + K - 2, \forall b_i \in B \quad (4)$$

We now prove that a YES instance of the NMTS-K problem occurs iff a YES instance of NMTS occurs. Every partition for the NMTS problem is associated with a partition for the NMTS-K problem. We denote the elements of  $X_i$  for  $i \geq 3$  as  $x_{ij}, 1 \leq j \leq m$  and let  $A_i$  be the partition for the NMTS problem. The associated partition for the NMTS-K problem  $A'_i$ , is defined as follows:  $A'_i = A_i \cup \{x_{ji} | 3 \leq j \leq K\}$ . This association immediately provides us with the equality:  $\sum_{x \in A'_i} s'(x) = (\sum_{x \in A_i} s(x)) + (K - 2)$  which we compare with the relation  $b'_i = b_i + (K - 2)$  from Eq. 4. We get that  $\sum_{x \in A'_i} s'(x) = b'_i$  and  $\sum_{x \in A_i} s(x) = b_i$  either happen simultaneously or not at all. Thus, this association ensures that this is a valid transformation.

The maximum number in the constructed instance is either the maximum size from  $X$  and  $Y$  or  $K - 2$  added to the maximum number from the target  $B$ ; both of which are polynomially bounded in the maximum integer in, and the length of, the NMTS instance. This, along with Remark 2.12 proves that NMTS-K is Strongly NP-Complete. ◀

### 3.1.2 K-PwT is Strongly NP-Complete

The K-PwT problem is Strongly NP-Complete by reduction from the NMTS-K problem. This problem can be regarded as a generalization of the 3-PARTITION problem where we are looking for a partition into  $K$ -sets and there are multiple targets to be reached instead of a single target.

► **PROBLEM 3.3 (K-PwT).** Given a set  $X$  with  $|X| = Km$ ,  $K \geq 2$ , a size function  $s : X \mapsto \mathbb{Z}^+$  and a target vector  $B = (b_1, \dots, b_m) \in \mathbb{N}^m$  with positive integer entries, can  $X$  be partitioned into  $m$  disjoint sets  $A_1, A_2, \dots, A_m$ , each containing exactly  $K$  elements, such that,  $\sum_{a \in A_i} s(a) = b_i$ , for  $1 \leq i \leq m$ ?

K-PwT  $(K, X, s, B, m)$  is an instance of the K-PwT problem characterized by the set  $X$ , an integer  $K$ , a size function  $s$ , the target vector  $B$  and the cardinality of the target vector  $m$ .

**Proof.** The K-PwT problem is in NP since given a particular candidate partition  $A_i$ , we only need to verify that,  $\sum_{a \in A_i} s(a) = b_i$ , for  $1 \leq i \leq m$ . We now construct an instance K-PwT  $(K', X, s', B', m')$  of K-PwT problem from an instance NMTS-K  $(K, X_i, s, B, m)$  of the NMTS-K problem in the following manner.

$M$  is polynomially bounded by the maximum integer in the NMTS-K instance.

$$M = KmM' \text{ where, } M' = \max(\{s(x_i) | x_i \in X_i\} \cup \{b_i | b_i \in B\}) \quad (5)$$

$$K' = K, X = \bigcup X_i \quad (6)$$

$$\text{For } 1 \leq i \leq K \text{ and } \forall x_j \in X_i : s'(x_j) = s(x_j) + M^i \quad (7)$$

$$B' = (b'_1, b'_2, \dots, b'_m) \text{ where } b'_j = b_j + \sigma, \forall b_j \in B \text{ and } \sigma = \sum_{i=1}^K M^i \quad (8)$$

The transformation is polynomial since equations 5 to 8 are polynomial-time computable. Now we show that a YES instance of the K-PwT problem occurs iff a YES instance of NMTS-K occurs. For ease of exposition, for the rest of the proof, we write all the numbers in K-PwT  $(K', X, s', B', m')$  in base  $M$ . We make three remarks, the first:  $\sigma$  is a  $K+1$  digit number with a 1 in all its digits except the rightmost or  $0^{th}$  digit (Eq. 8). The second, that every number  $s'(x)$  has a 1 as its  $i^{th}$  digit,  $s(x)$  in its rightmost digit<sup>2</sup> and 0 elsewhere. The third, a partition  $A_j$  for the NMTS-K instance  $(\bigcup X_i)$  is also a partition for the K-PwT instance  $(X)$ , irrespective of whether either of them solve the respective problems or not. We'll prove first that if  $A_j$  is a partition that solves the NMTS-K problem, then it also solves the K-PwT problem.

Let  $A_j$  be a partition that solves the NMTS-K problem. Using the same partition and Eq. 7 and 8, we get that  $\sum_{x \in A_j} s'(x) = \sum_{x \in A_j} s(x) + \sum_{i=1}^K M^i = \sum_{x \in A_j} s(x) + \sigma = b_j + \sigma = b'_j$ . This proves that if  $A_j$  solves the NMTS-K problem, then it also solves the K-PwT problem.

To prove the converse, let  $A_j$  be a partition that solves the K-PwT problem. We know that  $\sum_{x \in A_j} s'(x) = b_j + \sigma$ , which implies that  $\sum_{x \in A_j} s(x) + \sum_i \sum_{x \in X_i \cap A_j} M^i = b_j + \sigma$  (from Eq. 7). This in turn implies that  $\sum_{x \in A_j} s(x) = b_j$  and  $\sum_i \sum_{x \in X_i \cap A_j} M^i = \sigma = \sum_{i=1}^K M^i$  since  $s(x)$  does not contribute to  $\sigma$  (from the first two remarks and Eq. 8). Given  $\sum_i \sum_{x \in X_i \cap A_j} M^i = \sum_{i=1}^K M^i$ , equating the coefficients of the powers of  $M$ , we get that  $|X_i \cap A_j| = 1, \forall i, j$  which says that every set in the partition contains exactly one element from each of the sets  $X_i$ . We already know from the earlier equations that  $\sum_{x \in A_j} s(x) = b_j$ . Thus, the partition  $A_j$  is a solution to the NMTS-K problem as well.

The maximum integer in the K-PwT instance created by the transformation,  $\sigma + \max(b_1, \dots, b_m)$ , is bounded (from Eq. 5) by a polynomial in the maximum integer in, and the length of, the NMTS instance, which by Remark 2.12 makes this a pseudo-polynomial transformation. Thus, K-PwT is Strongly NP-Complete.  $\blacktriangleleft$

### 3.1.3 $E(I_S)$ is Strongly NP-Complete

We prove that the  $E(I_S)$  problem is Strongly NP-Complete by reduction from the K-PwT problem. We use a subclass of the K-PwT problem,  $\Pi_p$ , such that  $\text{MAX}[I] \leq p(\text{LENGTH}[I]), \forall I \in \Pi_p$ . By definition 2.10,  $\Pi_p$  is NP-Complete. The  $E(I_S)$  problem:

► **PROBLEM 3.4** ( $E(I_S)$ ). Given a sequence  $I_S$  does there exist a  $k$ -ary tree  $T$ , such that  $I_S = I_S(T)$ ?

The  $E(I_S)$  problem is characterized by the set  $S$  and the integer  $k$ . We refer to such an instance as  $E(I_S)(S, k)$ .

**Proof.** It is easy to see that the  $E(I_S)$  problem is in NP. Given  $T$ , one only needs to compare  $I_S$  with  $I_S(T)$  to see if the given tree realizes that sequence. We now construct an instance  $E(I_S)(S, k)$  from an instance of the K-PwT problem, K-PwT  $(K, X, s, B, m)$ , with  $k = K$ .

We define a number  $M$  which is a power of  $K$ , is much greater in magnitude than any of the other numbers in the problem, and is polynomially bounded by the maximum integer in the K-PwT instance<sup>3</sup> (Eq. 9 and 10). We also define  $m'$  and  $m''$  such that  $m + m'$  and

<sup>2</sup> Follows from Eq. 7 and  $M$  being much greater in magnitude than any number in the NMTS-K instance.

<sup>3</sup>  $K^{\lceil \log_K \alpha \rceil} \leq K^{1 + \log_K \alpha} = K\alpha$ .  $\therefore$  it is polynomially bounded.

$m + m''$  are powers of  $K$ . (Eq. 11).

$$M_1 = \max (\{s(x_i) | x_i \in X\} \cup \{b_i | b_i \in B\}) \quad (9)$$

$$M = K^{\lceil \log_K M_2 \rceil}, \text{ where } M_2 = KmM_1 \quad (10)$$

$$m' = K^d - Km, \quad m'' = K^{d-1} - m = m'/K, \text{ where } d = \lceil \log_K(Km) \rceil \quad (11)$$

We make the sequence  $S$  for the  $E(I_S)$  instance using four “component” sequences, namely the “child component”  $C$ , the “parent component”  $P$ , the “grandparent component”  $G$  and the “descendant component”  $D$ :

$$S = C \cup P \cup G \cup D \text{ where,} \quad (12)$$

$$C = C' \cup C'', \quad C' = \{s(x) + M \mid x \in X\}, \quad C'' = \{\overbrace{M, \dots, M}^{m' \text{ times}}\} \quad (13)$$

$$P = P' \cup P'', \quad P' = \{b_i + KM + 1 \mid b_i \in B\}, \quad P'' = \{\overbrace{KM + 1, \dots, KM + 1}^{m'' \text{ times}}\} \quad (14)$$

$$G = \bigcup_{i=0}^{d-2} l_i, \text{ where } l_i \text{ are “levels” defined later in the text.} \quad (15)$$

$$D = \bigcup_{i=1}^{K^d} D_i, \text{ where } D_i = \bigcup I_S(T^c(c_i)), \quad \forall c_i \in C \quad (16)$$

The “child component”  $C$  is the union of the sets  $C'$  and  $C''$ .  $C'$  is in one-to-one correspondence with the set  $X$ , using the sizes of elements from  $X$  with  $M$  added to them. The set  $C''$  is used to make the cardinality of set  $C$  to be a power of  $K$  using elements of value  $M$ . The “parent component”  $P$  is the union of the sets  $P'$  and  $P''$ .  $P'$  is in one-to-one-correspondence with  $B$  but has been modified to accommodate the changes made to sizes of elements of  $X$  while making  $C'$ .  $P''$  is used to make the cardinality of the set  $P$  to be a power of  $K$  using elements of value  $KM + 1$ .

We construct the “grandparent component” in “levels”. The lowest level  $l_{d-2}$  is constructed from  $P$ , by arbitrarily taking blocks of  $K$  elements, adding them all up and incrementing the result by one. Formally, we order the elements in  $P$  arbitrarily as  $P_1, P_2, \dots, P_{K^{d-1}}$  and then let  $l_{d-2} = \{l_{d-2,i} \mid l_{d-2,i} = 1 + \sum_{j=1}^K P_{(i-1)K+j}, 1 \leq i \leq K^{d-2}\}$ . Other levels  $l_{d-i}$  are constructed in a similar manner from levels  $l_{d-i+1}$ . This is continued until  $l_0$  which has only one element<sup>4</sup>. The element in  $l_0$  would be the largest number in the final instance.

The “descendant component” is constructed by using  $T^c(c_i)$ , the subtree sizes sequences of complete trees on each of the elements  $c_i \in C$  (Refer to definition 2.6). That is, for each such  $c_i$ , we make a complete  $k$ -ary tree on  $c_i$  nodes and find its subtree size sequence  $I_S(T^c(c_i))$ . Let these sequences be labeled  $D_i$ . The descendant component is  $D = \bigcup_i D_i = \bigcup_i I_S(T^c(c_i))$ .

This is a polynomial transformation since each element from the NMTS- $K$  instance is being used only once and each time a simple addition is done to get  $P$  and  $C$ . There are a logarithmic number ( $O(d)$ ) of  $l \in G$  and each is computed in polynomial time.  $D$  is made up of  $I_S(T^c(c_i))$  for each  $c_i \in C$  which has a polynomial number of elements and computing this for each element can be done in polynomial time<sup>5</sup>. Thus, every component can be computed

<sup>4</sup> The “parent component” was padded with elements until the number of elements became a power of  $K$ . Since at each level the number of elements gets reduced by exactly a factor of  $K$ , eventually exactly one element will remain.

<sup>5</sup> We are reducing from  $\Pi_p$ , which ensures that the elements in the  $C$  are polynomially bounded by  $\text{MAX}[I] \forall I \in \Pi_p$  and thus, the subtree sizes sequences on complete trees on these number of nodes can be computed polynomially.



in polynomial time and so the whole  $E(I_S)$  instance can be constructed in polynomial time.

Now we show that a YES instance of the NMTS-K problem occurs iff a YES instance of  $E(I_S)$  occurs. By construction, all elements in  $G$  and  $P$ , together, will form a k-ary tree with the elements in  $P$  as the leaves. Also  $C$  and  $D$  will make a forest of k-ary trees with each element from  $C$  as a root of one of the trees in the forest. Since  $C'$  is in one-to-one correspondence with  $X$  and  $P'$  is in one-to-one correspondence with  $B$ , if there is a partition,  $C'$  gets partitioned accordingly and these become the children of elements of  $P'$ .  $C''$  can be arbitrarily partitioned and made the children of elements of  $P''$ . This will provide the remaining edges to make a tree.

Now we need to prove that if there is a tree, then there is also a partition. For this, we only need to prove that the set of children of  $P'$  is equal to  $C'$ . To prove this, it is sufficient to show that the elements of  $P'$  and the elements of  $C'$  occur in consecutive levels in any tree. We use equations 13 to 16 to prove this. The element in  $l_0$  is the largest element and will necessarily have to be the root. This will be followed by the elements from  $l_1$  since no other elements are large enough to reach the element in  $l_0$ . Continuing this argument it is clear that the  $l_i \in G$  will always appear in consecutive levels in any tree and that  $P$  will follow immediately below these levels. Now, since no element from  $p \in P$  will be a child of any  $p' \in P$ , elements from either  $C$  or  $D$  will be needed to make child nodes of elements in  $P$ . But we note that elements in  $D$  will all be less than  $M/K$  in value which will not be enough to reach elements in  $P$  thus necessitating that all children of elements of  $P$  come from  $C$ . We note that elements from  $C'$  can not be children of elements from  $P''$  and so the set of children of  $P''$  will be equal to  $C''$ . Since a value of the order of  $KM$  has to be reached for elements in  $P'$  and all elements in  $C'$  are of the order of  $M$ , all the elements from  $C'$  will be used. Thus, if there is a tree, then it will have the elements from  $P'$  and  $C'$  in consecutive levels and therefore have a partition.

This transformation is sufficient to prove that  $E(I_S)$  is NP-Complete. For it to be Strongly NP-Complete, a subproblem of  $E(I_S)$  in which the maximum integer is bounded by the length of the instance has to be proven NP-Complete. We note that the maximum integer in this reduction is the element in  $l_0$ . We have already argued how this integer is polynomially bounded by the length of the problem. This in turn proves that this reduction is also sufficient to prove that  $E(I_S)$  is Strongly NP-Complete. ◀

► **Corollary 3.5** ( $E(I_S)$  for full k-ary trees). *The  $E(I_S)$  problem for full k-ary trees (existence of full k-ary trees given  $I_S$ ) is also Strongly NP-Complete.*

In the proof (Ref. Section 3.1.3) of the  $E(I_S)$  problem for k-ary trees, we are reducing (when it is a YES instance) the K-PWT instance to a full k-ary tree, except possibly in  $D$ .  $D$  is being made of complete trees on elements of  $C$ . A simple inductive proof is enough to show that changing every element  $c \in C$  to the form  $Kc + 1$  would be enough to make all of the complete trees in  $D$  to also be full trees. This (along with similar changes to  $P$ ) would allow the same reduction to be used to prove  $E(I_S)$  to be Strongly NP-Complete for full k-ary trees as well.

### 3.2 Sub-classes realizable in polynomial time

While the existence problems for a k-ary tree and for the full k-ary are Strongly NP-Complete, some sub-classes can be realized in polynomial time.

1. Complete Trees: The complete tree  $T^c(n)$  on a given number of nodes  $n$  has a unique structure. Given a sequence  $I_S$ , we construct  $T^c(|I_S|)$  and simply check if  $I_S = I_S(T^c)$ .



2. Degenerate Trees (A tree which is just a path):  $I_S$  must contain exactly one instance of every number from 1 to  $|I_S|$ .

These along with corollary 3.5 show that the structure of the tree we are trying to realize plays an important role in deciding the complexity of the problem. The full  $k$ -ary tree which allows flexibility in terms of the structure of the tree is Strongly NP-Complete, while for the more rigidly structured complete and degenerate trees, the problem is trivial.

## 4 Height and Depth

Depths, heights and subtree sizes can be recursively defined on the basis of the attribute values of some neighbor nodes and given a tree, each of the sequences  $I_S$ ,  $I_H$ ,  $I_D$  can be computed in linear time. We have seen that realizing trees given the subtree sizes sequence is NP-Complete but when we consider sequences of other attributes, we see that they are much easier to solve. We now provide  $O(n \log(n))$  algorithms for determining the existence of trees given height or depth sequences.

### 4.1 Depth

We know that in  $K$ -ary trees, every node can have at most  $K$  children. And by definition, depth values give the level at which a node is present. Let us define  $C_d$  to represent the number of times the value  $d$  occurs in the sequence  $I_D$ . If we have built a  $k$ -ary tree down to  $d$  levels then the next level can accommodate at most  $KC_d$  nodes. Hence, if for every  $d$  from 0 to  $d_{max} - 1$ , the condition  $C_{d+1} \leq KC_d$  holds, then a tree can be constructed. If the condition fails at any point then we get a proof that no tree exists.

$$C_{d+1} \leq KC_d, \text{ for } 0 \leq d \leq d_{max} - 1 \quad (17)$$

#### Sub-classes

- Complete tree: Eq. 17 is modified to be  $C_{d+1} = KC_d$  for all  $d \neq d_{max} - 1$ .
- Full  $K$ -ary tree: Along with the condition in Eq. 17 we add the extra condition that  $K$  divides  $C_d$  for all  $d$ . That is,  $K|C_d \quad \forall d$ .
- Degenerate Tree (A path): Exactly one count of each value must be present.

It is easy to see that these will all result in single pass algorithms once we have sorted the sequence and thus their running time is of the order of  $O(n \log(n))$  where  $n$  is the number of elements in the input sequence.

### 4.2 Height

Given a height sequence  $I_H$ , to solve the existence problem, we use the fact that, from the definition of height of a node, if a particular value  $h$  exists in the sequence then so would at least one instance of each value less than that  $(h - 1, h - 2, \dots, 2, 1)$ .

► **Definition 4.1 (Strand).** The strand of a node is a maximal path from the node to a leaf.

We divide the given sequence  $I_H$  into maximal strands by choosing greedily from the given sequence. We ensure that the maximal length strands are made before we make any of the smaller strands. So, first the root (the largest  $h \in I_H$ ) will get a strand of its own. Then, the next biggest remaining height value will get a strand and so on until no elements remain

in the sequence. This division is going to be unique because for a node to get a height value of  $h$ , it has to have at least one child whose height is  $h - 1$ , which in turn will need  $h - 2$  and so on. Thus, it gives us a necessary condition for the existence of a tree: if while making these strands, we get stuck at some point then no tree can be constructed.

Once these strands are constructed, then we connect them together to get a tree, if possible. Any strand with its root's height value as  $h$  can only be attached as a child of a node whose height value is at least  $h + 1$ . Thus if we just check if there are enough places where strands can be joined then we could answer whether a tree exists. This translates into a simple polynomial time algorithm stated in the following pseudo-code:

```
places:=0
count[i]:= number of nodes with height i
for i from h_max to 0:
    places := places + K*count[i] - count[i-1]
    if places < 0:
        No tree exists
Tree exists
```

The `places` variable tells us how many places are left at higher levels after each strand is added to the tree. If at some point, `places` becomes less than zero then the last strand we added was an invalid addition. Hence there would be no tree possible. If it is non-negative throughout then a tree would be possible.

#### Sub-classes

- Complete  $K$ -ary: Compare if  $I_H = I_H(T^c)$ , where  $T^c$  is the complete tree on  $|I_H|$  nodes.
- Full  $K$ -ary tree: If `places` is zero at the end of the loop, then a full  $K$ -ary tree is possible.
- Degenerate Tree: Exactly one count of each height value must be present.

## 5 Combining sequences

In this section we look at the problem of existence given sequences in combination.

### 5.1 Synchronized Height and Depth: $E(I_{HD})$

The algorithm for solving the  $E(I_{HD})$  problem combines ideas from the methods to solve the depth and height problems (Sections 4.1 and 4.2 respectively) and is easy to verify.

1. While finding maximal strands, ensure that depth values are also assigned in order.
2. Before computing `places`, put the roots of the strands at the correct depth.
3. At every level, check if dedicated possible parent exists for each strand rooted at that level. Do this in descending order of the root's height values.
4. If one gets stuck at any point in the algorithm, then no tree exists, otherwise, a tree exists.

**Asynchronized Height and Depth:  $E(I_H, I_D)$**  We do not know any solution to the asynchronized version of the height and depth sequences nor do we have a proof for NP-Completeness.

We have seen that realization given height and depth sequences be accomplished in polynomial time. We now attempt to solve the existence problem given subtree sizes by additionally providing depth and/or height sequences. Note that the existence of complete and degenerate trees can still be polynomially answered given any combination of depth and height sequences along with subtree sizes sequence (Section 3.2, 4.1 and 4.2).

► **Corollary 5.1.**  $E(I_S, I_D)$  and  $E(I_{SD})$  for  $k$ -ary as well as full  $k$ -ary trees are Strongly NP-Complete.

**Proof.** We show that the  $E(I_S)$  instance created during the reduction in Section 3.1.3 implicitly constructs instances of the  $E(I_S, I_D)$  and  $E(I_{SD})$  problems.

We note that the depth of the levels in  $G$ , are equal to the subscripts (0 to  $d - 2$ ) that are used. For  $P$  and  $C$  the depths are  $d - 1$  and  $d$ , respectively. The depths for the complete trees created in  $D$  can be computed along with the subtree sizes. Adding this information along with the subtree sizes information during reduction allows one to prove the  $E(I_S, I_D)$  and  $E(I_{SD})$  problems to be Strongly NP-Complete. Similar modifications to corollary 3.5 prove the Strongly NP-Completeness for full  $k$ -ary trees as well. ◀

► **Corollary 5.2.**  $E(I_S, I_H)$  and  $E(I_{SH})$  for  $k$ -ary as well as full  $k$ -ary trees are Strongly NP-Complete.

**Proof.** We show that the  $E(I_S)$  instance created during the reduction in Section 3.1.3 implicitly constructs instances of the  $E(I_S, I_H)$  and  $E(I_{SH})$  problems.

To know the height of a node, the heights of the children need to be known. Due to the way in which the child component is constructed (Eq. 13), a child component element  $c$  will always satisfy the following inequality  $K^h \leq c < K^{h+1}$  where  $h = \lceil \log(M_2) \rceil$ . This along with the fact that the descendant component attaches a complete subtree to  $c_i$  ensures that the heights of all  $c_i$  will be fixed at  $h$ . The parent and grandparent components' heights are decided based on the heights of the  $c_i$ , hence the heights of all the nodes in  $S$  can be computed during the transformation. Similarly with corollary 3.5 for full  $k$ -ary versions. ◀

► **Corollary 5.3.** Both the synchronized and asynchronous sequences of heights, depths and subtree sizes are NP-Complete. (Obvious from Corollaries 5.1 and 5.2.)

Since the height and depth sequences are not enough to provide a solutions, we look at providing the inorder traversal rank (the position of a node during inorder traversal) synchronized with subtree sizes; we denote this as  $I_{S, ITR}$  and the problem as  $E(I_{S, ITR})$ . While this can be extended to  $k$ -ary trees, it is most easily illustrated in binary trees where the inorder traversal rank is the rank of the node as it would have been in a binary search tree. Since we know the root, it allows for trivial partitioning of the remaining elements into either the left or the right subtree. After that step, one is left with two smaller problems. This would allow one to use a divide-and-conquer method to get a polynomial time solution; giving us the following corollary:

► **Corollary 5.4.**  $E(I_{S, ITR})$  can be solved in polynomial time.

The  $E(I_{S, ITR})$  problem throws light into the structural difficulty in the  $E(I_S)$  problem. In the  $E(I_S)$  problem, say for binary trees, deciding the root node's subtree size is obvious (the largest value in the sequence, say  $r$ ). After that, the second largest number, say  $l$ , is necessarily one of the children (let it be the left child) and  $r - l$  will be the remaining child of the root node. So, the root and its two children can be easily decided (if they exist) but after that, partitioning the remaining nodes into the left or right subtree of the root is a difficult task. This is like the NP-Complete PARTITION problem but is more complex since not only does the partition have to sum up to a particular value, the partition must also be realizable as a tree.

Adding the inorder traversal ranks allows definitive partitioning of the nodes into left/right subtrees. This is then no longer similar to the PARTITION problem, leading to a polynomial

time solution. This shows that the PARTITION like nature of the  $E(I_S)$  problem is an important part of what makes it Strongly NP-Complete. We also note that whenever we are able to solve a variant of the  $E(I_S)$ , we are always creating a unique tree. As soon as there is structural flexibility in the construction of the tree, the problem becomes difficult.

## 6 Conclusion

In this paper we look at the problem of the existence of rooted k-ary trees given some combination of sequences of attributes like subtree sizes, heights and depths. We prove the problem of the existence of the tree given the subtree sizes sequence to be Strongly NP-Complete; problems that additionally provide height and/or depth sequences, in either synchronized or asynchronized manner still have the same complexity. We also prove that in each of these cases, when asked about the existence of full k-ary trees, the problem remains Strongly NP-Complete. Existence of trees given  $I_H$ ,  $I_D$  and  $I_{H,D}$  can be solved in polynomial time, even the problem of existence of full k-ary trees. For all of these problems existence of complete and degenerate trees can be solved in polynomial time.

Apart from these, when the inorder traversal rank is given synchronized with the subtree sizes sequence, the existence of a tree can be answered in polynomial time. We argued that this is evidence that the difficulty in the problem is in its partitioning like nature. We also argued, by comparing the complexity of complete and degenerate tree variants with the full tree variant, that the uncertainty or freedom in the structure of a tree plays a role in the intractability of the problem.

There are many areas for future work. For problems related to subtree size sequences, exact exponential, approximation algorithms or other such strategies can be searched for, which allow solving it more efficiently. One could also look for minimal super-sequences or maximal sub-sequences which realize a tree. The contrasting nature of the subtree sizes, height and depth attributes can be studied to gain a better understanding of the relation between them in terms of realizability. Along the same lines, we have not been able to find either an algorithm or a proof of NP-Completeness for the asynchronized height and depth sequence problem. Studying this problem might throw insight into how the height and depth of k-ary trees are related. Our focus throughout the paper has been on rooted k-ary trees but all these problems can also be asked for general rooted trees.

**Acknowledgments.** The author would like to thank Professor Rahul Muthu and Professor Srikrishnan Divakaran for their continued guidance and for their helpful suggestions. The author also thanks Professor Jayanth Varma for his review of the draft.

---

## References

- 1 Jayadev Acharya, Hirakendu Das, Olgica Milenkovic, Alon Orlitsky, and Shengjun Pan. String reconstruction from substring compositions. *SIAM Journal on Discrete Mathematics*, 29(3):1340–1371, 2015.
- 2 Georgy Maximovic Adelson-Velskii and Evgenii Mikhailovich Landis. An information organization algorithm. In *Doklady Akademii Nauk SSSR*, volume 146, pages 263–266, 1962.
- 3 Martin Aigner and Eberhard Triesch. Realizability and uniqueness in graphs. *Discrete Mathematics*, 136(1-3):3–20, 1994.
- 4 Dénes Bartha and Péter Burcsi. Reconstructibility of trees from subtree size frequencies. *Studia Universitatis Babes-Bolyai, Mathematica*, 59(4), 2014.

- 5 T. Cormen, C. Leiserson, R. Rivest, and Stein C. *Introduction to algorithms*. MIT press, 2009.
- 6 Miroslav Dudík and Leonard J Schulman. Reconstruction from subsequences. *Journal of Combinatorial Theory, Series A*, 103(2):337–348, 2003.
- 7 P Erdos and T Gallai. Graphs with points of prescribed degrees. *Mat. Lapok*, 11(264-274):132, 1960.
- 8 Michael R Garey and David S Johnson. “Strong”NP-Completeness Results: Motivation, Examples, and Implications. *Journal of the ACM (JACM)*, 25(3):499–508, 1978.
- 9 Michael R Garey and David S Johnson. Computers and Intractability: A guide to the theory of NP-completeness. *WH Free. Co., San Fr*, 1979.
- 10 Leo J Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science*, pages 8–21. IEEE, 1978.
- 11 S Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph I. *Journal of the Society for Industrial and Applied Mathematics*, 10(3):496–506, 1962.
- 12 Frank Harary. On the reconstruction of a graph from a collection of subgraphs. In *Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963)*, pages 47–52, 1964.
- 13 Frank Harary. A survey of the reconstruction conjecture. In *Graphs and Combinatorics*, pages 18–28. Springer, 1974.
- 14 Frank Harary and Ed Palmer. On the problem of reconstructing a tournament from sub-tournaments. *Monatshefte für Mathematik*, 71(1):14–23, 1967.
- 15 Václav Havel. A remark on the existence of finite graphs. *Casopis Pest. Mat*, 80(477-480):1253, 1955.
- 16 Paul J Kelly et al. A congruence theorem for trees. *Pacific J. Math*, 7(1):961–968, 1957.
- 17 László Lovász. A note on the line reconstruction problem. In *Classic Papers in Combinatorics*, pages 451–452. Springer, 2009.
- 18 Bennet Manvel. Reconstruction of trees. *Canad. J. Math*, 22:55–60, 1970.
- 19 Bennet Manvel. On reconstructing graphs from their sets of subgraphs. *Journal of Combinatorial Theory, Series B*, 21(2):156–165, 1976.
- 20 Bennet Manvel and Paul K Stockmeyer. On reconstruction of matrices. *Mathematics Magazine*, 44(4):218–221, 1971.
- 21 C Nash-Williams. Reconstruction of locally finite connected graphs with at least three infinite wings. *Journal of graph theory*, 11(4):497–505, 1987.
- 22 Kaushal Shah. Rooted tree with the given subtree sizes. Unpublished manuscript from Bachelor Project done at DAIICT, Gandhinagar, 2014.
- 23 S.M. Ulam. A collection of mathematical problems. *Wiley*, 1960.